WHAT IS CLAIMED IS:

1. A processor comprising:

an out-of-order microinstruction pointer (μIP) stack in a microcode (μcode) execution core.

2. The processor of claim 1 in which the μIP stack comprises:

an entry number field;

a microinstruction pointer (μIP) field;

a back pointer field;

a retirement indicator field; and

a return pointer field.

3. The processor of claim 2 in which the μIP field is 14-bits wide.

4. The processor of claim 3 in which the μIP field has a microinstruction pointer (μIP) pushed by a first microoperation (μOp) code and used by a second μOp code.

5. The processor of claim 2 in which the back pointer field has a pointer to a next entry in the μIP stack for a micro-type of service (μTOS) bit to point to after a μOp.

6. The processor of claim 2 in which the retirement indicator field has an indication of whether an entry has retired.

7. The processor of claim 2 in the return pointer field a pointer to a location in a retirement stack to which an entry is copied after being retired.

8. A method executed in a processor comprising:

executing microcode (μcode) stored in an out-of-order microinstruction pointer (μIP) stack; and

manipulating the μIP stack with a set of microinstructions.

9. The method of claim 8 in which the stack has an entry number field, a microinstruction pointer (μIP) field, a back pointer field, a retirement indicator field and a return pointer field.

10. The method of claim 9 in which the μIP pointer field is 14-bits wide.

11. The method of claim 10 in which the μIP pointer field has a microinstruction pointer (μIP) pushed by a first microoperation (μOp) code and used by a second μOp code.

12. The method of claim 9 in which the back pointer field has a pointer to a next entry in the μIP stack for a micro-type of service (μTOS) bit to point to after a μOp.

13. The method of claim 9 in which the retirement indicator field has an indication of whether an entry has retired.

14. The method of claim 9 in which the return pointer field contains a pointer to a location in a retirement stack to which an entry is copied after being retired.

15. The method of claim 9 in which manipulating comprises:

pushing a next μIP on to the μIP stack; and

using the next μIP in an intermediate field as a target μIP in a jump operation.

16. The method of claim 9 in which manipulating comprises:

taking a value of an intermediate field of a microoperation (μOp); and

pushing the value on to the μIP stack.

17. The method of claim 9 in which manipulating comprises:

popping a value off the μIP stack; and

replacing a current μOp intermediate field.

18. The method of claim 9 in which manipulating comprises:

popping a value off of the μIP stack; and

jumping to that value.

19. The method of claim 9 in which manipulating comprises:

reading a value off the μIP stack; and

replacing a μOp's intermediate field with the value.

20. The method of claim 9 in which manipulating comprises setting the μIP stack pointers to reset.

21. The method of claim 9 further comprising providing a set of pointers that point to different entries in the μIP stack.

22. The method of claim 21 in which the set of pointers includes a μTOS pointer that points to a top of the μIP stack.

23. The method of claim 21 in which the set of pointers includes a μAlloc pointer that points to a next allocated entry in the μIP stack.

24. The method of claim 21 in which the set of pointers includes a NextRet pointer that points to a next entry in the μIP stack to be deallocated.

25. The method of claim 21 in which the set of pointers includes μRetTos pointer that points at a retired top of the μIP stack.

26. The method of claim 8 in which the μOPs include an ms_call μOP that takes a next μIP, pushes the next μIP on the μIP stack, and uses the next μIP in an intermediate field as a target μIP of a jump.

27. The method of claim 8 in which the μOPs include an ms_push μOP that takes a value in an intermediate field and pushes the value on the μIP stack.

28. The method of claim 8 in which the μOPs include an ms_pop μOP that pops a value off the μIP stack and replaces the value with the μOP's intermediate field.

29. The method of claim 8 in which the μOPs include an ms_return μOP that pops a value off of the μIP stack and jumps to that μIP.

30. The method of claim 8 in which the μOPs include an ms_tos_read μOP that reads a value off the μIP stack and replaces this μOP's intermediate field.

31. The method of claim 8 in which the μOPs include an ms_μip_stack_clear μOP that sets the μIP stack pointers to reset.

32. A computer program product residing on a computer readable medium having instructions stored thereon which, when executed by the processor, cause the processor to:

execute microcode (μcode) stored in an out-of-order microinstruction pointer (μIP) stack; and

manipulate the μIP stack with a set of microinstructions.

33. The computer program product of claim 32 wherein instructions to manipulate further comprise instructions to:

push a next μIP on to the μIP stack; and

use the next μIP in an intermediate field as a target μIP in a jump operation.

34. The computer program product of claim 32 wherein instructions to manipulate further comprise instructions to:

take a value of an intermediate field of a microoperation (μOp); and

push the value on to the μIP stack.

35. The computer program product of claim 32 wherein instructions to manipulate further comprise instructions to:

pop a value off the μIP stack; and

replace a current μOp intermediate field with the value.

36. The computer program product of claim 32 wherein instructions to manipulate further comprise instructions to:

pop a value off of the μIP stack; and

jump to that value.

37. The computer program product of claim 32 wherein instructions to manipulate further comprise instructions to:

read a value off the μIP stack; and

replace a μOp's intermediate field with the value.

38. The computer program product of claim 32 wherein instructions to manipulate further comprise instructions to:

set the μIP stack pointers to reset.